Technology Consulting Company
Research, Development &
Global Standard

PREEMPT_RTの移植
－ 進捗報告 －

松原 克弥
株式会社イーゲル

Funded by 株式会社ルネサスソリューションズ

CE Linux Forum

背景

igel

- ユーザレベルでデバイスドライバを実現したい
  - 開発が容易
  - ドライババグによるシステムダウンを軽減
  - より密接なアプリとの連携
- いくつかの問題
  - I/Oメモリ、物理メモリへのアクセス
  - 割り込み要求(IRQ)の受信
  - 応答速度
  - ・・
- カーネル2.6の新機能
  - NPTL(Native POSIX Thread Library)
  - スケジューラの改善(O(1)スケジューラ等)
  - カーネルプリエンプション
  - ・・

これまでの過程

igel

1. スレッドの応答性調査
2. ULDD(User Level Device Driver)フレームワークの設計
3. ULDDのプロトタイプ実装
4. Linuxリアルタイム対応の調査

今日は、
PREEMPT_RTを適用した2.6.16カーネルを
Renesas SH4ベースのボード
RTS7751R2D への移植作業を報告します。

4. Linuxのリアルタイム対応

igel

- 完全リアルタイム
  - 全タスクのデッドラインを保証 ⇒ (非リアルタイム)既存OSへの接木では困難
  - Nested OS, Dual OS/Dual Core, ART-Linux
- 部分リアルタイム
  - 一部のタスクのデッドラインを努力保証
  - たいていのリアルタイムタスクは限られた資源のみを使用する。

  ⇒遅延を最小に ＝プリエンプション

  ⇒ 遅延度合いをより小さく ＝ リアルタイムプリエンプション
  - CONFIG_PREEMPT, CONFIG_PREEMPT_RT

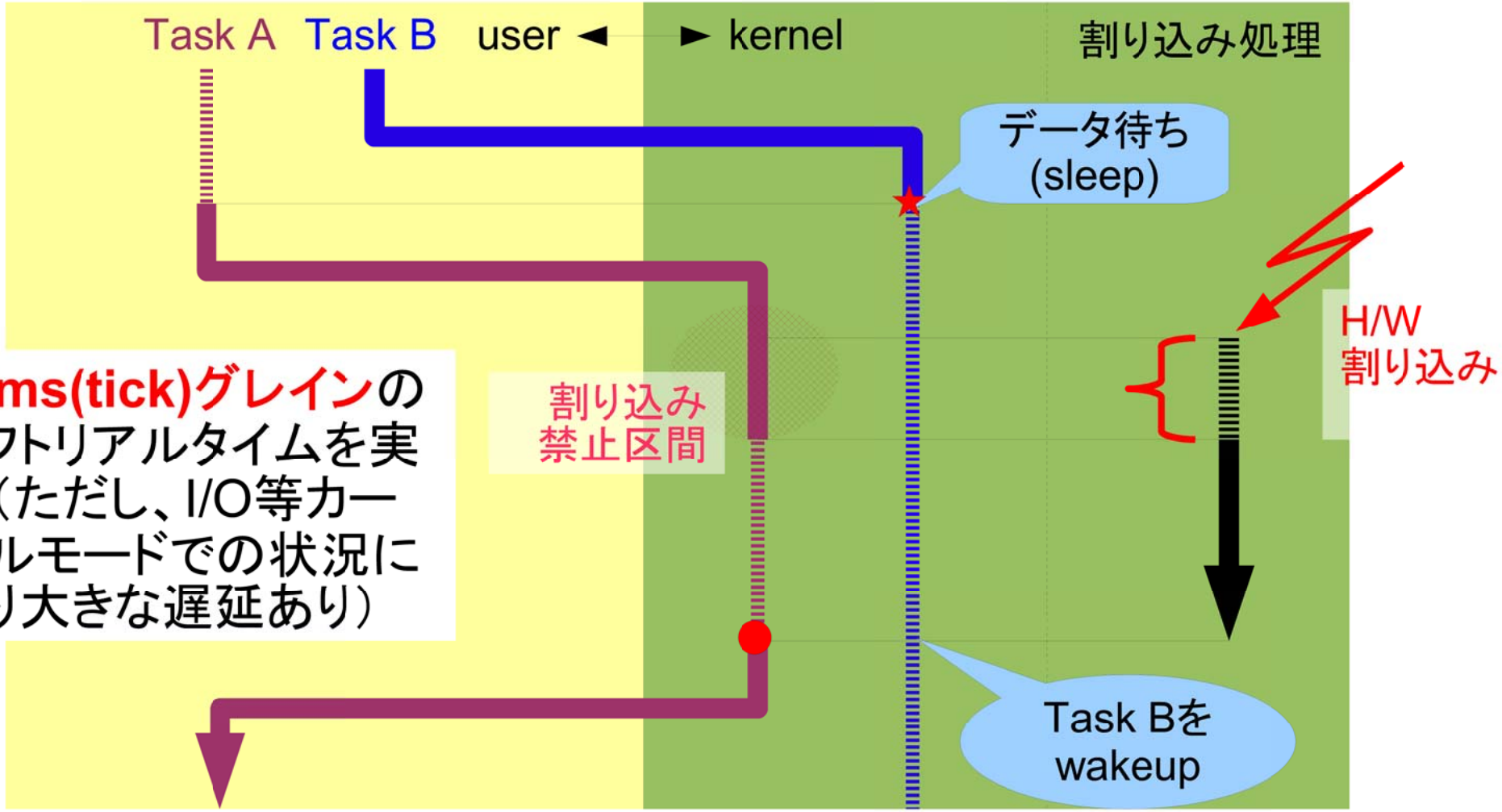
Linuxのプリエンプション

igel

- プリエンプションのタイミング
  - 割り込み処理からの復帰時
  - システムコールからの復帰時
  - タスクが自主的に休眠した時

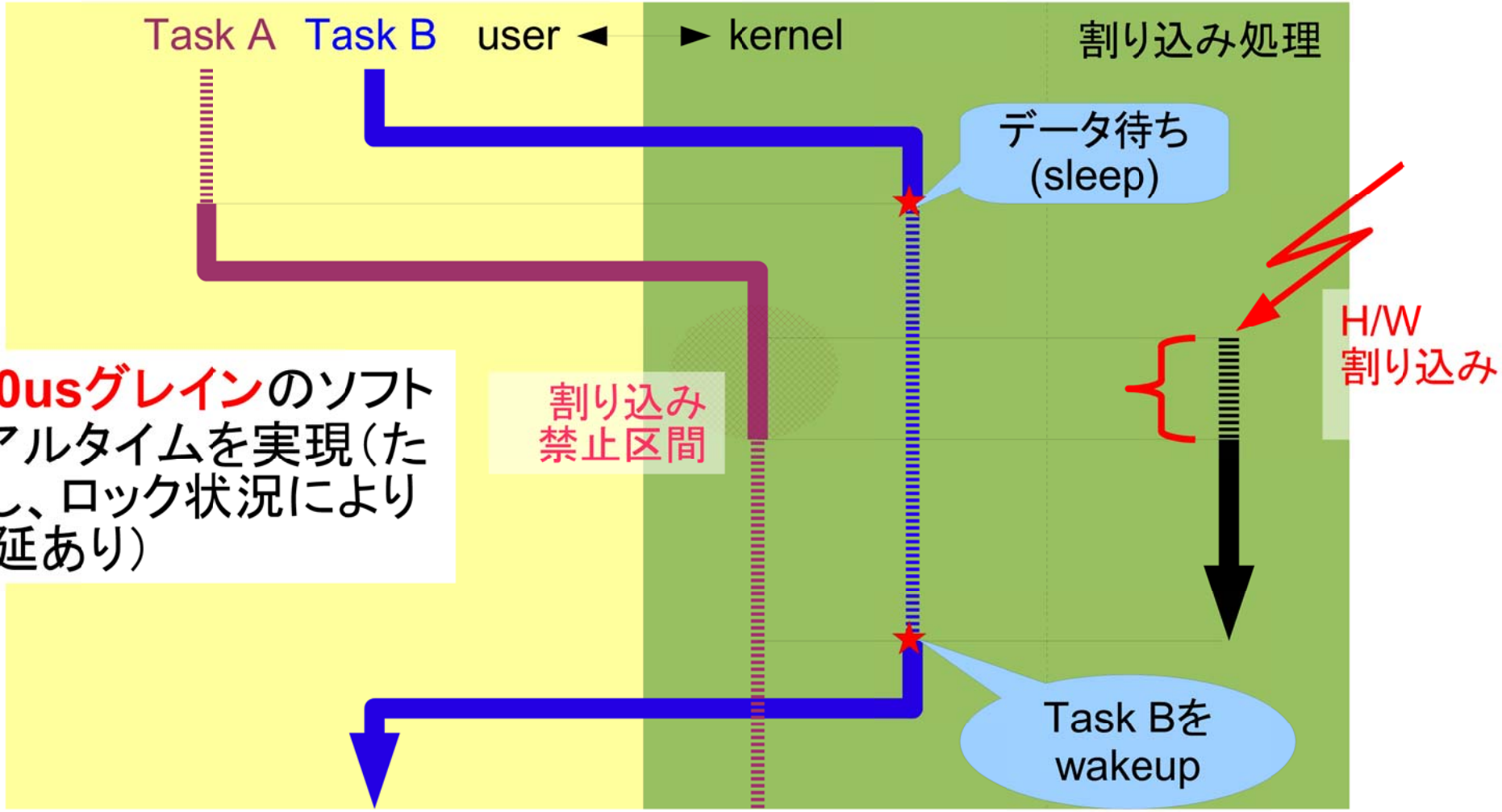
  ⇒リターン時に別タスクのスタックポインタをセットし、pop
- 2.4 / CONFIG_PREEMPTなしの2.6では、カーネルモードではプリエンプション(タスク切り替え)が起きない(割り込み処理は起きる)
  - カーネルモード時の割り込みからの復帰は元の場所に戻る(プリエンプション非対応コードのため)

2.4 / 2.6 without CONFIG_PREEMPT



10ms(tick)グレインの
ソフトリアルタイムを実
現（ただし、I/O等カー
ネルモードでの状況に
より大きな遅延あり）

CONFIG_PREEMPT



100usグレインのソフトリアルタイムを実現(ただし、ロック状況により遅延あり)

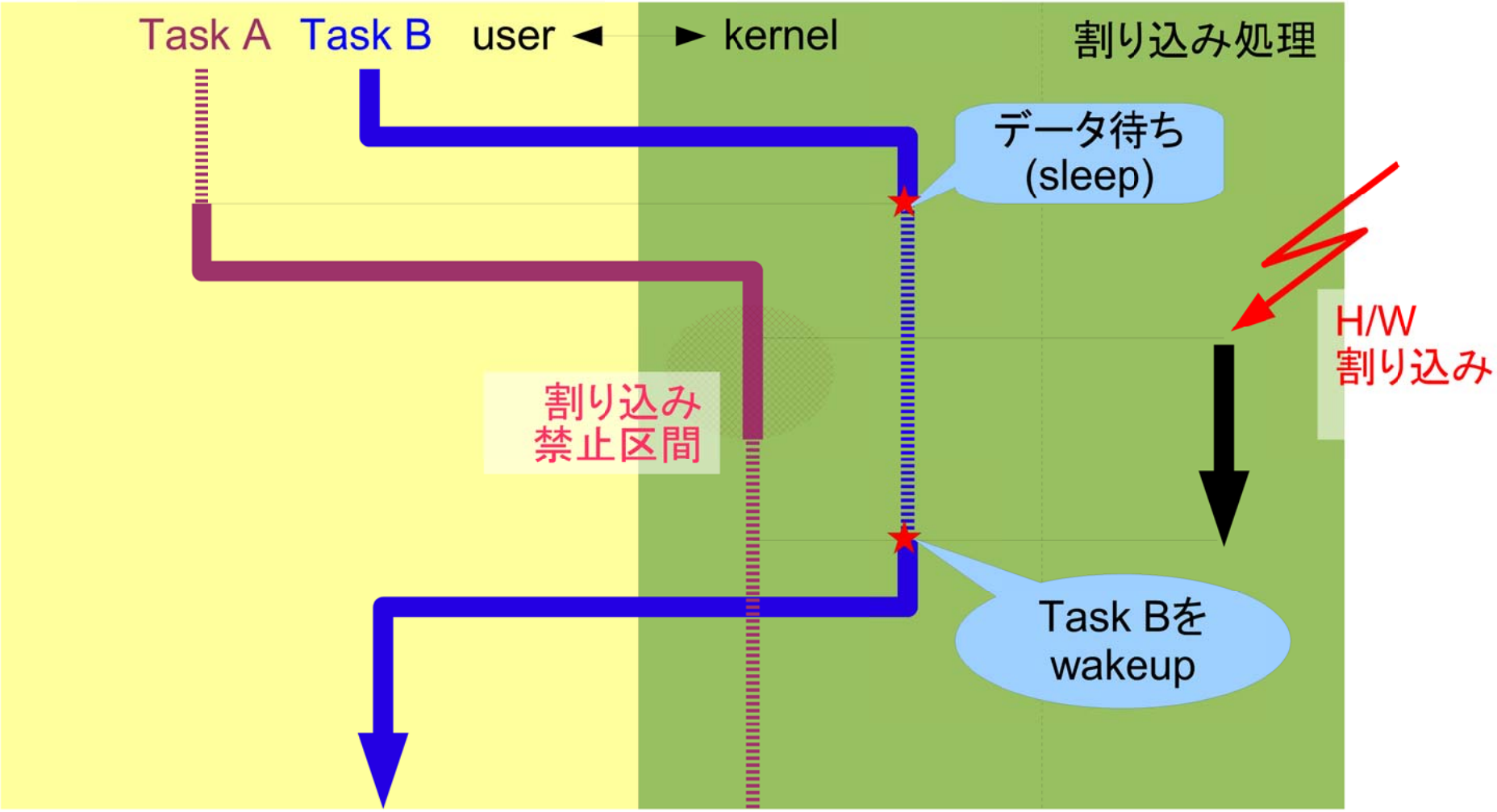
(CONFIG_)PREEMPT_RT

igel

Ingo MolnarらのLinux realtime preemptionパッチ

- プリエンプションの機会を、割込み処理中やクリティカル領域内にも増やすことで遅延を軽減し、タスク起床および割り込み処理において、20～30usのソフトリアルタイムを実現
- 優先度逆転を防ぐため、優先度継承ロックを提供
- 他コード部分のリファクタリング・最適化
- http://people.redhat.com/mingo/realtime-preempt/

PREEMPT_RT


Task A
Task B
user ◄
► kernel
割り込み処理
データ待ち
(sleep)
H/W
割り込み
割り込み
禁止区間
Task Bを
wakeup

PREEMPT_RT により追加・変更される特徴

igel

i. クリティカル領域内でのプリエンプション
ii. 割り込み処理内でのプリエンプション
iii. 割り込み禁止コード領域内でのプリエンプション
iv. スピンロックとセマフォの優先度継承
v. その他の最適化

PREEMPT_RTの移植

igel

- 現在サポートされているアーキテクチャ
  - Intel x86, PowerPC, ARM, MIPS等
- 新しいアーキテクチャへの対応
  - arch/*アーキテクチャ*、include/asm-*アーキテクチャ*内の変更
    - 割り込み禁止、プリエンプション禁止にすべきロックやセマフォに「raw_」「compat_」を付与
  - (あまり使われていない)デバイスドライバの変更
    - 割り込みハンドラがプロセスコンテキストでも実行可能か
    - ロックや割り込み禁止箇所の見直し
  - タイマハンドラ等の割り込みコンテキストで実行すべき割り込みハンドラの登録にSA_NODELAYを指定
  - スケジューラ呼び出しに関連する処理部分を見直し
  - その他、アーキテクチャ別(if defined(*アーキテクチャ*))の処理を追加

PREEMPT_RTの適用手順

igel

1. kernel.orgからカーネルソースを取得、展開
2. RTパッチを適用
3. .configの変更
4. アーキテクチャ依存コード、デバイスドライバ・コードの見直し
5. コンパイルして動作（安定するまで4.に戻る）

1. カーネルソースの展開

igel

- これは説明しなくてもよいですね。(^^;

- Tips: 2.6.x.yにはRTパッチがあたらないことが多いです。
- 必要に応じて、ターゲット用パッチ（例：linuxsh）を適用

2. RTパッチの適用

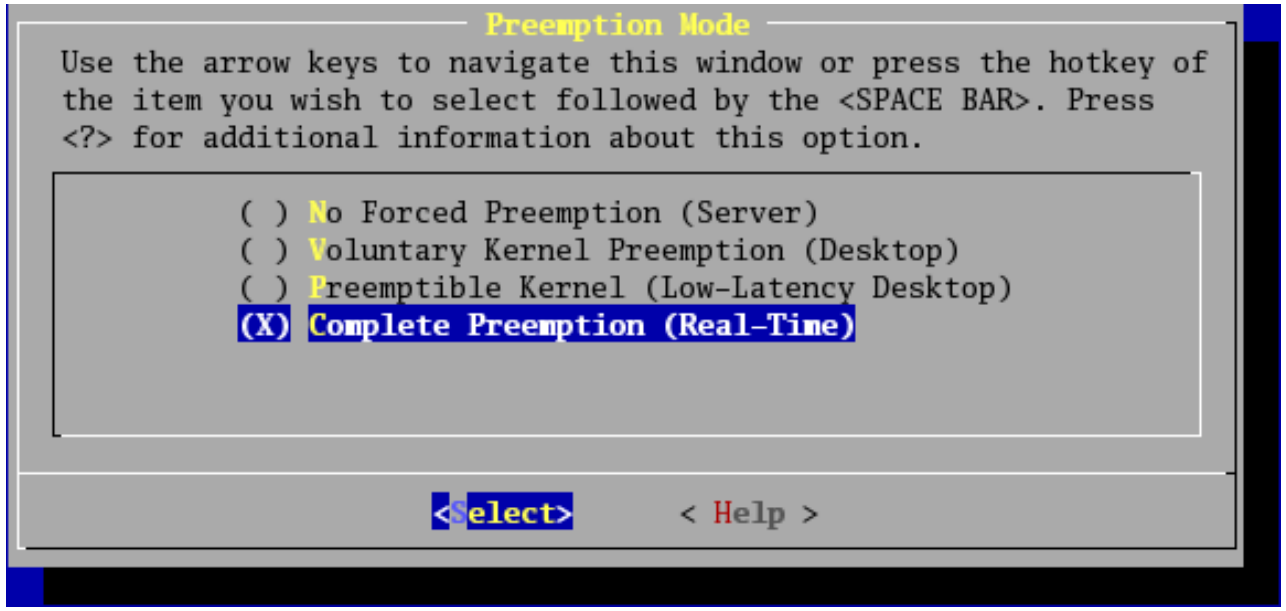
igel

- IngoのWebページからパッチをダウンロード
  - http://people.redhat.com/mingo/realtime-preempt/
- 181のディレクトリ（親子関係を含む）下にある687のファイルへのパッチ（2.6.16-rt23の場合）

3. .configの変更

igel

■ PREEMPT_RTを有効にする。



4. ターゲット依存コードの再確認

igel

ここが一番やっかい・・

- 下記処理の振る舞いが変わるため、アーキテクチャ依存コード（arch/, include/asm-*/）およびデバイスドライバ（drivers/）のコードを見直す必要がある。
  - スピンロック
  - 割り込み禁止
  - 割り込みハンドラ
  - スケジューラに関わる処理部分等

スピンロックの再検討

igel

- スピンロックがスリープする。
→使用箇所でスリープしても問題ないかを確認する必要あり

- 見直しが必要な例
  - 割り込みハンドラ内のスピンロック
  - 割り込み禁止中 or プリエンプション禁止中のスピンロック
  - スピンロックで作られたクリティカルリージョン内のアトミック処理

割り込み禁止処理の再検討

igel

- 割り込み禁止が禁止にならない。

→本当に割り込みを禁止しなければならない箇所は変更する必要あり

- 見直しが必要な例
  - アトミック処理を実現するために用いられている割り込み禁止
  - 割り込みハンドラが動かないことを保証するために使われている割り込み禁止

割り込みハンドラの再検討

igel

- 割り込みハンドラがプロセスコンテキストで動く。

→ハンドラ処理内で遅延とプリエンプションが許容できるかを判定する必要がある。

- 見直しが必要な例
  - 排他制御が期待されている処理
  - プリエンプション（＆さらなる割り込み）がないことを期待されている処理
  - （大きな）遅延が許されない処理

例:include/asm-sh/atomic.h

igel

```
25 static __inline__ void atomic_add(int i, atomic_t * v)
26 {
27         unsigned long flags;
28
29         local_irq_save(flags);
30         *(long *)v += i;
31         local_irq_restore(flags);
32 }
33
34 static __inline__ void atomic_sub(int i, atomic_t *v)
35 {
36         unsigned long flags;
37
38         local_irq_save(flags);
39         *(long *)v -= i;
40         local_irq_restore(flags);
41 }
```

- PREEMPT_RTでは割り込みが禁止されない。
  - 割り込みハンドラとの間でatomic変数を共有している場合は、まずい。
    →**raw_**local_irq_save/restore()に変更

5. コンパイルして動作確認

igel

- これも説明はいらないですね。
- 安定して動かない場合は、コード見直し＆動作確認を繰り返すことに･･。

RTS7751R2Dでの作業

igel

- Linux 2.6.16
  + linuxsh patch
  + RTS7751R2D用パッチ
  + patch-2.6.16-rt29
  - アーキテクチャ依存コードの書き換え（Thanks リネオさん）
  - 不必要なraw_, compat_の削除と追加
  - デバイスドライバ・コードの見直し
  - その他不都合回避

余談:get_wchan() でOops

igel

- コードを見てみると・・

```
441 unsigned long get_wchan(struct task_struct *p)
442 {
443         unsigned long schedule_frame;
444         unsigned long pc;
445
446         printk(KERN_DEBUG "%s: 1¥n", __FUNCTION__);
447         if (!p || p == current || p->state == TASK_RUNNING)
448                 return 0;
449
450         /*
451          * The same comment as on the Alpha applies here, too ...
452          */
453         printk(KERN_DEBUG "%s: 2¥n", __FUNCTION__);
454         pc = thread_saved_pc(p);
```

- procfsからのみコールされている＆psコマンドでしか使わなさそうなので、常に0を返して問題回避（テンポラリ）

get_wchan() for alpha

igel

- きれいなコードではないらしいが・・プリエンプションとの関係は不明

```
492 unsigned long
493 get_wchan(struct task_struct *p)
494 {
495         unsigned long schedule_frame;
496         unsigned long pc;
497         if (!p || p == current || p->state == TASK_RUNNING)
498                 return 0;
499         /*
500          * This one depends on the frame size of schedule().  Do a
501          * "disass schedule" in gdb to find the frame size.  Also, the
502          * code assumes that sleep_on() follows immediately after
503          * interruptible_sleep_on() and that add_timer() follows
504          * immediately after interruptible_sleep().  Ugly, isn't it?
505          * Maybe adding a wchan field to task_struct would be better,
506          * after all...
507          */
508
509         pc = thread_saved_pc(p);
```

あっ、動作したっ！・・かも？

igel

- 「安定」の判定が難しい。
  - クリティカルリージョンや割り込みハンドラの振る舞いが変わるので、発見しにくいレースコンディションやデッドロックを見つけなければならない。
  - すべてのデバイスを高負荷状態で評価する？
  - 「安定な方向へ倒しすぎ」も判定しにくい。
- 「効果」の判定が難しい。
  - 低負荷状態では恩恵が見えにくい。さらには、スループット低下もありうる。
  - 移植の方法（プリエンプションの機会を減らす）によっては、効果がないことも考えられる。

  ←レイテンシの改善はもちろんだが、通常時のスループット低下も評価したい。

まとめとTODO

igel

- PREEMPT_RTをSuperHボード上に移植しました（しています）。

- パッチを公開してたたいてもらう。
- 他のボード（デバイス）に対応させる。
- Ingo Molnar or LKMLになげて、RTパッチにマージしてもらう？
- ULDDへの効果を評価する。